Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 182 (2007) 107–122

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

SAT-based Abstraction Refinement for Real-time Systems

Stephanie Kemper[1,3]

Centrum voor Wiskunde en Informatica, Amsterdam

André Platzer[2,4]

University of Oldenburg, Department of Computing Science, Germany
Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

In this paper, we present an abstraction refinement approach for model checking safety properties of real-time systems using SAT-solving. We present a faithful embedding of bounded model checking for systems of timed automata into propositional logic with linear arithmetic and prove correctness. With this logical representation, we achieve a linear-size representation of parallel composition and introduce a quick abstraction technique that works uniformly for clocks, events, and states. When necessary, abstractions are refined by analysing spurious counterexamples using a promising extension of counterexample-guided abstraction refinement with syntactic information about Craig interpolants. To support generalisations, our overall approach identifies the algebraic and logical principles required for logic-based abstraction refinement.

Keywords: abstraction refinement, model checking, real-time systems, SAT, Craig interpolation

1 Introduction

Failures within embedded systems of automotive industry, railway technology, and avionics usually have disastrous consequences. One dominant feature of these safety-critical systems is that safety crucially depends on reactions that occur *in time*. For instance, a train controller needs to apply the brakes as a response to driving faster than the current track situation permits. Yet, this response has to be executed in time before reaching an open gate or it will be useless.

[1] Part of this research has been funded by the Dutch BSIK/BRICKS project.

[2] This work was partially supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS, see `www.avacs.org`) and by a fellowship of the German Academic Exchange Service (DAAD).

[3] Email: `s.kemper@cwi.nl`

[4] Email: `platzer@informatik.uni-oldenburg.de`

doi:10.1016/j.entcs.2006.09.034

The computational complexity introduced by the infinite state space of these *real-time systems* leads to severe limitations in scalability even within very well-established model checkers like Uppaal (http://www.uppaal.com). Aside from the omniscient state explosion problem [4] already present in finite state model checking, current model checking techniques for real-time systems are still limited in the number of concurrent quantitative temporal observations (measured by clocks). A particularly dramatic cause of the state explosion problem is the exponential blow-up obtained by forming the cross product for parallel composition of timed automata (TA). To avoid this, we define a linear-size parallel composition for the logical representation of TA. Typically, only a reduced part of the full parallel composition has to be expanded from our representation during satisfiability checking (SAT solving).

Very sophisticated and well-optimised techniques (e.g., [10]) guide high-end SAT solvers to explore only a comparably narrow fragment around the part of the state space relevant for the particular safety property. We build upon this development by choosing a linear arithmetic/propositional encoding (as opposed to implementing new algorithms for the restricted domain of TA from scratch): a philosophy that has successfully proven its great potential in finite state systems [3]. With this basis, we exploit the particularities of transition systems induced by timed automata using abstraction refinement to deal with the challenges of infinite states.

Timed Automata

As the prevailing model for real-time systems, TA [1] are an extension of finite automata with real-valued variables (clocks), that can measure the passing of time. Their behaviour consists of a sequence of events happening over time. TA allow two types of events: visible (external) and invisible (internal) actions. The former are used for synchronisation with other automata, while the latter are used for internal steps of a single automaton, independent from others. Transitions may reset clocks and are considered to be instantaneous, i.e., time may only elapse while the automaton remains in one of its states. The firing of transitions and the dwell time are states is restricted by clock constraints—called guards and invariants, respectively—which the current clock values have to satisfy.

The AVACS Project

The project AVACS (Automatic Verification and Analysis of Complex Systems) addresses the rigorous mathematical analysis of models of complex safety critical computerised systems, such as aircrafts, trains, cars, or other artifacts, whose potential failures can endanger human life. In this context, we have implemented a prototypical model checker in Java, called SAAtRe (SAT-based Abstraction Refinement). As a SAT solver backend, SAAtRe uses FOCI[5], since it is able to derive interpolants, which play an important role in the refinement step in Section 5.

[5] By Kenneth McMillan (based on [9]).

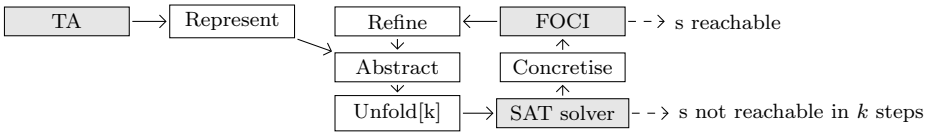


Fig. 1. SAT-based Abstraction Refinement Loop

Structure of the Abstraction Refinement Loop

Abstraction refinement [4,6] is a promising direction of research to overcome the challenges of the state explosion problem and infinite state model checking, while preserving correctness of verification results. Abstraction techniques over-approximate system behaviour by removing constraints that are considered irrelevant for verifying a particular specification. If the abstract system is safe (no error state is reachable) then, by conservative over-approximation, so is the original.

Fig. 1 depicts a conceptual overview of the data-flow in our abstraction refinement loop (grey boxes represent external tools). We **represent** the transition characteristics of a TA in propositional logic with linear arithmetic, and automatically produce a simpler **abstract** version of it. After **unfolding** the resulting transition formula (k times), a satisfiability check solves the bounded reachability question in the abstract system. Depending on the outcome, the real-time system has either been proven safe (s is unreachable) within bound k, or needs to be analysed with respect to an abstract counterexample (**concretised**), again using SAT solving (FOCI). If the abstract counterexample has a counterpart in the non-abstracted system, then the real-time system is unsafe. Otherwise, the counterexample is *spurious* and results from an inappropriate initial choice of abstraction. Analysing the counterexample (with Craig interpolants derived by FOCI) then helps to **refine** the abstraction and start over until the system is proven safe or unsafe.

Related Work

Audemard *et al.* [2] reduce bounded model checking for TA to the satisfiability problem of a mathematical formula. They elaborate on encoding LTL specifications of TA. Without abstraction techniques, scalability of this approach is still limited, though.

Jhala and McMillan [7] present an abstraction refinement approach for predicate abstraction. Using interpolants, they generate refinements which take into account specific characteristics of the property. A limitation, however, is the fact that their approach relies on an appropriate choice of predicates for predicate abstraction. Our approach can be considered as a quick (hence, scalable) approximation of predicate abstraction, where predicate discovery is evident by exploiting the nature of TA.

The abstraction refinement framework presented by Clarke *et al.* [4] works with Kripke structures originating from finite state programs. In contrast, our approach deals with the challenges of infinite state model checking as introduced by the notion of real-time clocks. Further, we directly use a formula representation tailored for SAT-based bounded model checking.

Structure of this Paper

After introducing real-time systems and bounded model checking in Section 2, we present a faithful representation of TA in propositional logic with linear arithmetic for bounded model checking in Section 3, and give a soundness result. In Section 4, we introduce a uniform abstraction, and extend the algebraic perspective on soundness from Section 3 to correspondence results about abstraction. Section 5 closes the abstraction refinement loop by investigating how spurious counterexamples can be exploited for refining abstractions, before concluding with a summary and future work in Section 6.

2 Preliminaries

In this section we introduce standard notions for TA [1] and bounded model checking [3], and present our running example.

Timed Automata

Definition 2.1 [Timed automata] Let $\mathcal{TA}$ be the set of all *timed automata* $\mathcal{A} = (\underline{A}, S, s_0, X, I, E)$ with

- $\underline{A}$ is a finite set of *(visible) events*, with typical elements a, a';
- S is a finite set of *states*, with *initial state* $s_0 \in S$ and typical elements s, s';
- X is a finite set of (real-valued) *clocks*, with typical elements x, y;
- $I : S \to \Phi(X)$ is a mapping that assigns an *invariant* to each state; and
- $E \subseteq S \times (\underline{A} \cup \{\tau\}) \times \Phi(X) \times \mathcal{P}(X) \times S$ is a set of *transitions*; with (s, a, φ, Y, s') denoting an action transition from s to s' on occurrence of event a, restricted by the *guard* φ and resetting all clocks in the set Y.

Clock constraints (i.e., guards and invariants) $\varphi \in \Phi(X)$ are formulas of propositional logic with clock comparisons, $x - y \sim c$ and $x \sim c$ for $c \in \mathbb{Q}$, $\sim \in \{>, \geq, <, \leq, =\}$, and the usual connectives $\neg, \wedge, \vee$. A clock valuation ν is a mapping assigning a real value to every clock, which represents the time elapsed since the corresponding clock was last reset. To ensure that invariants hold amongst subsequent steps, they are assumed convex (i.e., do not contain $\vee, \neg$ [1]), which is an important property used for efficient representation (Section 3.2). The special *invisible* event $\tau \notin \underline{A}$ denotes an internal action that happens without communication with another automaton.

The trace semantics, $Trace_{\mathcal{A}}$, of a TA $\mathcal{A}$ is defined as the set of all traces of the associated transition system $\mathcal{S}_{\mathcal{A}}$ [1]: a *configuration* (s, ν) of $\mathcal{S}_{\mathcal{A}}$ consists of a state $s \in S$ and a clock valuation ν that satisfies the invariant $I(s)$. Transitions of $\mathcal{S}_{\mathcal{A}}$ reflect the way that $\mathcal{A}$ may evolve in time: a *delay transition* $(s, \nu) \xrightarrow{\delta} (s, \nu + \delta)$ increases all clock values by the same amount of time δ, an *action transition* $(s, \nu) \xrightarrow{a} (s', \nu')$ resets the clocks in Y to zero on the event a and leaves the others unchanged while following an edge $(s, a, \varphi, Y, s') \in E$ on which the guard φ is true for ν. Time has to increase monotonically without converging (non-Zeno traces). $Trace_{\mathcal{A},k}$ is defined as the set of prefixes of a trace of $\mathcal{A}$ with (at most) length k.

The combination of multiple automata performing joint broadcast communication (note that [1] uses pairwise synchronisation) forms a parallel real-time system; its semantics is defined as that of the corresponding product automaton (we present a technique to avoid the exponential cross product in Section 3.4).

Definition 2.2 [Product of TA] Let $\mathcal{A}=(\underline{A}, S, s_0, X, I, E)$ and $\tilde{\mathcal{A}}=(\underline{\tilde{A}}, \tilde{S}, \tilde{s_0}, \tilde{X}, \tilde{I}, \tilde{E})$ be TA with $X \cap \tilde{X} = \emptyset$. The *product of* $\mathcal{A}$ *and* $\tilde{\mathcal{A}}$ is the TA $\mathcal{A}\|\tilde{\mathcal{A}}$, defined as $\mathcal{A}\|\tilde{\mathcal{A}} = (\underline{A} \cup \underline{\tilde{A}}, S \times \tilde{S}, (s_0, \tilde{s_0}), X \cup \tilde{X}, I_\|, E_\|)$ with

- $I_\|(s, \tilde{s}) = I(s) \wedge \tilde{I}(\tilde{s})$ and
- Transitions $(s, a, \varphi, Y, s') \in E$ and $(\tilde{s}, \tilde{a}, \tilde{\varphi}, \tilde{Y}, \tilde{s}') \in \tilde{E}$ give rise to:
  - $((s, \tilde{s}), a, \varphi, Y, (s', \tilde{s})) \in E_\|$ iff $a \notin \underline{\tilde{A}}$ or $a = \tau$ (discrete transition of $\mathcal{A}$),
  - $((s, \tilde{s}), \tilde{a}, \tilde{\varphi}, \tilde{Y}, (s, \tilde{s}')) \in E_\|$ iff $\tilde{a} \notin \underline{A}$ or $\tilde{a} = \tau$ (discrete transition of $\tilde{\mathcal{A}}$),
  - $((s, \tilde{s}), a, \varphi \wedge \tilde{\varphi}, Y \cup \tilde{Y}, (s', \tilde{s}')) \in E_\|$ iff $a = \tilde{a}$ (synchronisation).

Bounded Model Checking for Timed Automata

Bounded model checking (BMC) has turned out to be amongst the most promising approaches for verification of safety properties [3]. The principle is to examine prefix fragments of the transition system, and successively increase the exploration bound until it reaches (a computable indicator of) the diameter of the system—or an unsafe trace has been discovered.

Definition 2.3 [Bounded safety] Let $\mathcal{A}=(\underline{A}, S, s_0, X, I, E)$ be a TA, let $s \in S$ be an error state. $\mathcal{A}$ is *safe with respect to* s *within bound* k, denoted by $\mathcal{A} \models_k \neg\mathbf{EF}s$, if there is no finite trace $t_k \in Trace_{\mathcal{A},k}$, starting from s_0 and ending in s. Otherwise, $\mathcal{A}$ is called *unsafe with respect to* s.

On the basis of these $\neg\mathbf{EF}s$ reachability properties, other bounded LTL specifications can be verified using the encoding in [2].

Running Example

Fig. 2 shows a slightly adapted version of the well-known Train-Gate-Controller example (see, e.g., [1]): trains have an additional state ("emergency stop") which they enter in case the controller detects they are approaching too fast ($v<2$). Trains may cross the gate only after they received a "go" signal, which is sent by the controller after the gate has closed succesfully. The error state of this system is $in \wedge \neg down$, i.e., a train crosses a gate that is (partially) open.

3 Representation of Timed Automata

In the sequel, let $\mathcal{A}=(\underline{A}, S, s_0, X, I, E)$ be a TA. In this section, we construct a formula, $\varphi(\mathcal{A})$, in propositional logic with linear arithmetic that represents the transition relation of $\mathcal{A}$, defined in terms of the transition characteristics from step $t-1$ to step t. Unfolding the resulting transition formula k times yields a variant $\varphi(\mathcal{A})_k$,

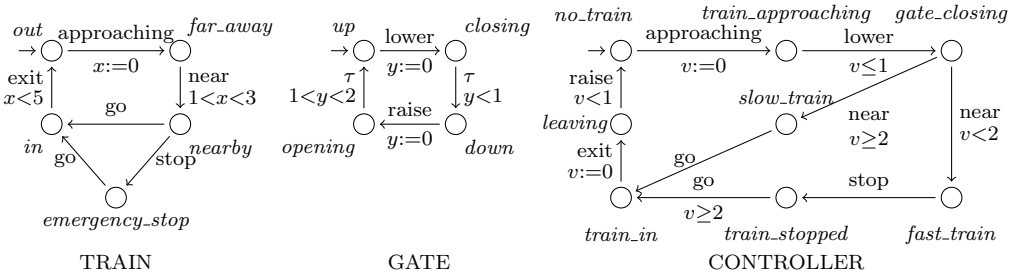


Fig. 2. Train-Gate-Controller example

which represents all possible behaviour of $\mathcal{A}$ for the first k steps (see Section 3.3). This formula, together with a representation of the safety property, is unsatisfiable iff $\mathcal{A}$ is safe within bound k.

3.1 Overview: The Fundamental Concepts

The possible behaviour of a TA depends on the current system configuration (i.e., state and clock valuation). Since these change with time, the truth of a formula ψ about the corresponding configuration of a TA depends on the time step t (see [11] for details). Hence, we define ψ_t as the *localisation* of ψ, which is obtained by adding the index t to all propositional letters and variable symbols occurring in ψ. Thus, if ψ is of vocabulary s, x, a, then ψ_t will refer to $\mathtt{s}_t, \mathtt{x}_t, \mathtt{a}_t$ instead. In particular:

States For every state $s \in S$, the Boolean variable $\mathtt{s}_t$ represents whether the automaton is in state s at step t.

Events For every event $a \in \underline{A} \cup \{\tau\}$, the Boolean variable $\mathtt{a}_t$ represents whether the automaton executes a transition in step t that is labelled with a.

Clocks For every clock $x \in X$, the rational variable $\mathtt{x}_t$ (*clock reference*) represents the absolute point in time when x was last reset up to step t. An additional rational variable $\mathtt{z}_t$ (*absolute time reference*) represents the absolute amount of time that has passed until step t. The *clock value* of clock x at step t is thus obtained by $\mathtt{z}_t - \mathtt{x}_t$. This temporal difference representation significantly improves the SAT solving performance, due to the decreased number of arithmetic operations [8].

Observe that even though clocks are real-valued, a rational encoding is sufficient since linear arithmetic is equisatisfiable for rational and real variables [8].

3.2 Transition Relation

The representation of the transition relation has to model both action and delay transitions. It constrains the possible valuations of variables representing the automaton configuration at subsequent step t depending on those at $t-1$.

Definition 3.1 [Timed automaton representation] For a TA $\mathcal{A}$, the formula representation $\varphi(\mathcal{A})$ of its transition relation in propositional logic with linear arithmetic is defined in equation (8) of Fig. 3.

$$e_a(s,a,\varphi,Y,s') \stackrel{def}{=} \mathtt{s}_{t-1} \wedge \mathtt{s}'_t \wedge \mathtt{a}_{t-1} \wedge \varphi_{t-1} \wedge (\mathtt{z}_{t-1} = \mathtt{z}_t) \bigwedge_{x \notin Y} (\mathtt{x}_{t-1} = \mathtt{x}_t) \wedge \bigwedge_{x \in Y} (\mathtt{x}_t = \mathtt{z}_t) \quad (1)$$

$$e_d(s) \stackrel{def}{=} \mathtt{s}_{t-1} \wedge \mathtt{s}_t \wedge (\mathtt{z}_{t-1} \leq \mathtt{z}_t) \wedge \bigwedge_{x \in X} \mathtt{x}_{t-1} = \mathtt{x}_t \wedge \bigwedge_{a \in \underline{A}} \neg \mathtt{a}_{t-1} \quad (2)$$

$$\varphi_E(\mathcal{A}) \stackrel{def}{=} \bigvee_{(s,a,\varphi,Y,s') \in E} e_a(s,a,\varphi,Y,s') \vee \bigvee_{s \in S} e_d(s) \quad (3)$$

$$\varphi_i(\mathcal{A}) \stackrel{def}{=} (\mathtt{s}_0)_0 \wedge \bigwedge_{s_0 \neq \tilde{s} \in S} (\neg \tilde{\mathtt{s}}_0) \wedge (\mathtt{z}_0 = 0) \wedge \bigwedge_{x \in X} (\mathtt{x}_0 = 0) \quad (4)$$

$$\varphi_S(\mathcal{A}) \stackrel{def}{=} \bigwedge_{s,s' \in S, s \prec s'} \neg(\mathtt{s}_t \wedge \mathtt{s}'_t) \quad (5)$$

$$\varphi_{\underline{A}}(\mathcal{A}) \stackrel{def}{=} \bigwedge_{a,a' \in \underline{A} \cup \{\tau\}, a \prec a'} \neg(\mathtt{a}_{t-1} \wedge \mathtt{a}'_{t-1}) \quad (6)$$

$$\varphi_I(\mathcal{A}) \stackrel{def}{=} \bigwedge_{s \in S} (\neg \mathtt{s}_{t-1} \vee I(s)_{t-1}) \quad (7)$$

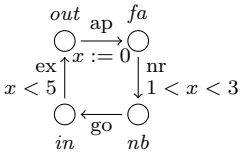
$$\varphi(\mathcal{A}) \stackrel{def}{=} \varphi_S(\mathcal{A}) \wedge \varphi_{\underline{A}}(\mathcal{A}) \wedge \varphi_E(\mathcal{A}) \wedge \varphi_i(\mathcal{A}) \wedge \varphi_I(\mathcal{A}) \quad (8)$$

$$\varphi(\mathcal{A})_k \stackrel{def}{=} \bigwedge_{1 \leq j \leq k} \varphi(\mathcal{A})_{j/t} \quad (9)$$

Fig. 3. Representation (8) and k-unfolding (9) of TA: transitions (1), (2), (3), initial conditions (4), mutual exclusion constraints (5), (6), invariants (7).

Before executing an action transition $(s,a,\varphi,Y,s') \in E$ of step t in (1), the automaton is in state s (at step $t-1$), event a occurs and guard φ is satisfied. The values of the absolute time reference and clock references of $X \setminus Y$ do not change, all clock references in Y are adjusted to the actual point in time. After the execution (at step t), the automaton is in state s'. For a delay transition (2), the automaton remains in state s, the value of the absolute time reference increases, all clock references keep their value (the time of last reset does not change), and no event $a \in \underline{A}$ must occur. Due to convexity, the invariant only needs to be checked at the end of the time delay (it inductively holds at the beginning (4)). The disjunction of these formulas expresses a (nondeterministic) transition choice (3). The automaton starts in its initial state s_0, with all initial clock values equal to zero (4). Furthermore, the automaton is in only one state at a time (mutual exclusion of state variables (5)) with at most one event (6) ($\prec$ denotes an arbitrary but fixed order on S and $\underline{A} \cup \{\tau\}$, respectively), to prevent $\varphi(\mathcal{A})$ from following multiple transitions simultaneously. The invariant of the actual state has to hold at any time ((7), where $I(s')_t$ denotes the localisation of $I(s')$, cf. Section 3.1).

Example 3.2 [Representation] Consider the four transitions of the TRAIN automaton depicted in Fig. 4(a) (with names abbreviated). The representation of these transitions is shown in Fig. 4(b).



(a) Transitions

$$
\begin{array}{l}
(\mathtt{out}_{t-1} \wedge \mathtt{fa}_t \wedge \mathtt{ap}_{t-1} \qquad\qquad\qquad\qquad \wedge (\mathtt{z}_{t-1}{=}\mathtt{z}_t) \wedge (\mathtt{x}_t{=}\mathtt{z}_t)) \\
\vee (\mathtt{fa}_{t-1} \wedge \mathtt{nb}_t \wedge \mathtt{nr}_{t-1} \wedge (1{<}\mathtt{z}_{t-1}{-}\mathtt{x}_{t-1}{<}3) \wedge (\mathtt{z}_{t-1}{=}\mathtt{z}_t) \wedge (\mathtt{x}_{t-1}{=}\mathtt{x}_t)) \\
\vee (\mathtt{nb}_{t-1} \wedge \mathtt{in}_t \wedge \mathtt{go}_{t-1} \qquad\qquad\qquad\quad \wedge (\mathtt{z}_{t-1}{=}\mathtt{z}_t) \wedge (\mathtt{x}_{t-1}{=}\mathtt{x}_t)) \\
\vee (\mathtt{in}_{t-1} \wedge \mathtt{out}_t \wedge \mathtt{ex}_{t-1} \wedge (\mathtt{z}_{t-1}{-}\mathtt{x}_{t-1}{<}5) \quad \wedge (\mathtt{z}_{t-1}{=}\mathtt{z}_t) \wedge (\mathtt{x}_{t-1}{=}\mathtt{x}_t))
\end{array}
$$

(b) Transition Representation

Fig. 4. Representation: Running Example

3.3 Unfolding for Bounded Model Checking

In order to represent the reachability problem of BMC in logic, the formula representation $\varphi(\mathcal{A})$ is unfolded, i.e., instantiated for all steps 1 to bound k. The resulting formula $\varphi(\mathcal{A})_k$ is called *k-unfolding of $\mathcal{A}$*, and defined in (9) ($\psi_{j/t}$ denotes the localisation of ψ—cf. Section 3.1—with t replaced by j).

Intuitively, a satisfying interpretation (or *model*) of $\varphi(\mathcal{A})_k$ corresponds to a trace of $\mathcal{A}$ of length k, i.e., to one possible behaviour of $\mathcal{A}$ for the first k steps (Section 3.6). BMC amounts to conjoining $\varphi(\mathcal{A})_k$ with $\rho_k \stackrel{def}{=} s_0 \vee s_1 \vee \ldots \vee s_k$, then $\mathcal{A} \models_k \neg \mathbf{EF} s$ holds iff the conjunction is unsatisfiable.

3.4 Parallel Systems

In this section, we introduce a linear-size logical representation of systems of TA without forming the exponential cross product (Def. 2.2).

Within a parallel real-time system, automata perform joint broadcast synchronisation on visible events ($\neq\tau$): if event a occurs, every automaton $\mathcal{A}_i$ with $a \in \underline{A}_i$ ("knowing about a") has to execute a transition labelled with a, or do a zero-delay step ("nothing") if $a \notin \mathcal{A}$. If, instead, event τ occurs, automata may decide to execute either a transition labelled with τ or to do a zero-delay step. Delay steps with delay $d{>}0$ have to be executed synchronously by all automata. The product representation is just the conjunction of the individual representations, where (6) is understood to be defined globally for the union of all events, since at any point in time, only one event may occur.

3.5 Discussion

Propositional formulas as an intermediate representation have several advantages: most importantly, our approach can use high-performance SAT solving technology for verification. Secondly, it generalises easily to other transition-based systems like hybrid automata [5]. Once a translation into formulas ("frontend") is defined, the same abstraction refinement framework ("backend") can be reused.

Moreover, our representation is specifically tailored for SAT solving technology. In addition to providing conjunctive normal form (CNF) whenever possible, observe that (5) and (6) are binary clauses, which are very efficient (the 2-SAT problem is polynomial since binary clauses do not increase the breadth of the search space). Formula (7) also corresponds to a set of binary clauses, whereas (4) gives rise to unit clauses. Note that it is immediate to adapt our representation to logarithmic

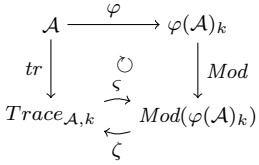


Fig. 5. Correctness of Representation

encoding of states and events. Using an encoding of addition with carry in propositional logic, mutual exclusion can be expressed with a linear number of clauses by saying that true state variables add up to one. However, despite this theoretical advantage, it is not obvious which variant to prefer in practice, since—unlike the linear representation—(5) restricts to binary clauses. While by the disjunctive nature of transition choices, (3) is not in CNF, it can be directly transformed to short CNF when introducing new symbols. For parallel systems, our logical representation avoids the exponential product construction and is linear in the number of automata.

3.6 Correctness

For the representation $\varphi(\mathcal{A})$ to be faithful (i.e., exhibit the same behaviour as $\mathcal{A}$), every model of $\varphi(\mathcal{A})_k$ has to correspond to a trace of length k, and vice versa. This is captured formally in the following theorem (see [8] for a formal proof):

Theorem 3.3 (Correctness of representation) *The TA representation is correct, i.e., the diagram in Fig. 5 commutes*[6] *.*

Here, the commutative property expresses that models of $\varphi(\mathcal{A})_k$ have a bijective correspondence to traces of the original TA $\mathcal{A}$, denoted by correspondance maps ζ and ς: the trace $\zeta(\varsigma(t))$ of the model $\varsigma(t)$ belonging to some trace $t \in Trace_{\mathcal{A},k}$ again is t (i.e., $\zeta(\varsigma(t)) = t$), and the model $\varsigma(\zeta(m))$ of a trace $\zeta(t)$ belonging to some model $m \in Mod(\varphi(\mathcal{A})_k)$ again is m (i.e., $\varsigma(\zeta(\sigma)) = \sigma$).

4 Abstraction

In this section, we present a simple, fast but nevertheless powerful uniform abstraction technique specifically tailored to work on logical formulas: *abstraction by merging omission (MO)*. By removing constraints which are considered irrelevant to the particular safety property, MO yields an over-approximation.

4.1 Abstraction by Merging Omission

The basic idea of MO is to reduce the system complexity by decreasing the number of symbols in $\varphi(\mathcal{A})$ while retaining as much information about the transition characteristics as possible (the abstract formula is weaker than $\varphi(\mathcal{A})$, though). It is

[6] A diagram is commutative [12] iff, between each two nodes in the diagram, following every path yields the same result. For instance, $Mod(\varphi(\mathcal{A})) = \varsigma(tr(\mathcal{A}))$ or $\zeta(Mod(\varphi(\mathcal{A}))) = tr(\mathcal{A})$.

defined for formulas in negation normal form (NNF), which already holds for $\varphi(\mathcal{A})$ (guards can be transformed to NNF easily). In the sequel, let $\mathcal{A}=(\underline{A}, S, s_0, X, I, E)$ a TA, let $\Sigma=\underline{A}\cup S$ the set of propositional letters (states and events, without indices); the set of real variables (clock references) is X. To illustrate the effect of abstraction on the different syntactical categories, we define it in terms of a state event mapping γ (called *map of merging*) on Σ and of a clock abstraction set $\mathcal{AS}$ (called *set of omission*).

Definition 4.1 [Abstraction by merging omission] Let F a formula in NNF with propositional letters in Σ and real variables in X. Let $\mathcal{AS}\subseteq\Phi(X)\cup X$ a set not containing compound formulas, and let $\gamma{:}\Sigma\to\Sigma'$ a mapping to some set Σ' of propositional letters, with $\Sigma\cap\Sigma'=\emptyset$. The *abstraction by merging omission* of F with respect to $\mathcal{AS}$ and γ is defined by applying transformation α; it is depicted in Fig. 6.

$$\alpha(L) = \begin{cases} L & \text{if } \mathit{cont}(L)\cap(\mathcal{AS}\cup\Sigma)=\emptyset \\ \gamma(L) & \text{if } \mathit{cont}(L)\cap\Sigma\neq\emptyset, L \text{ positive} \\ \texttt{true} & \text{otherwise} \end{cases} \tag{10}$$

$$\alpha(F\wedge G)=\alpha(F)\wedge\alpha(G)$$
$$\alpha(F\vee G)=\alpha(F)\vee\alpha(G)$$

Here, F and G are formulas in NNF, L is a literal, and $\mathit{cont}(L)$ is the set of atomic formulas and variables occurring in L.

Fig. 6. Abstraction by Merging Omission

Lifting α to the presence of localisations is straightforward: γ and $\mathcal{AS}$ are understood oblivious to indices in the NNF of $\varphi(\mathcal{A})$, such that indices directly carry over to $\varphi(\mathcal{A})_k$ unchanged (defining different abstractions for different steps is possible using the same definition of α but we consider it to be less useful).

MO uniformly captures abstraction on propositional letters and on real variables or atomic predicates about these (and thus is able to abstract all symbols defined in Section 3.1). The map γ is the identity for symbols not meant to be abstracted. States (or events) with the same image, however, are merged. In this way, α performs a quick variant of existential abstraction [4], but exploits the structural relationships of clocks and TA. Further, α is homomorphic with respect to $\{\wedge,\vee\}$, which proves the equality of $\alpha(\varphi(\mathcal{A})_k)$ and $\alpha(\varphi(\mathcal{A}))_k$ (except for speed of computing the abstraction, where $\alpha(\varphi(\mathcal{A}))_k$ is superior).

Observe that—unlike negative clock constraints—α always maps negative propositional letters to $\texttt{true}$. In general, this is necessary, since $\neg\mathtt{s}$ does not allow to conclude $\neg\mathtt{u}$ in case of a merge $\gamma(r)=\gamma(s)=u$. For our setting, it is more efficient to regenerate those parts of $\varphi(\mathcal{A})_k$ that contain negative propositional variables (in particular (5) and (6)) after applying α, anyway.

4.2 Running Example

Consider the formula presented in Fig. 4(b). If $x \in \mathcal{AS}$, MO substitutes all (atomic) subformulas containing x by `true`, and the formula finally simplifies to

$$\begin{array}{rl} & (\mathtt{out}_{t-1} \wedge \mathtt{fa}_t \wedge \mathtt{ap}_{t-1} \wedge (\mathtt{z}_{t-1} = \mathtt{z}_t)) \vee (\mathtt{nb}_{t-1} \wedge \mathtt{in}_t \wedge \mathtt{go}_{t-1} \wedge (\mathtt{z}_{t-1} = \mathtt{z}_t)) \\ \vee & (\mathtt{fa}_{t-1} \wedge \mathtt{nb}_t \wedge \mathtt{nr}_{t-1} \wedge (\mathtt{z}_{t-1} = \mathtt{z}_t)) \vee (\mathtt{in}_{t-1} \wedge \mathtt{out}_t \wedge \mathtt{ex}_{t-1} \wedge (\mathtt{z}_{t-1} = \mathtt{z}_t)) \end{array}$$

Note that even though clock x is abstracted, it imposes restrictions on traces which can be retained: between subsequent events *approach* and *exit*, clock values may increase by at most 5. This information can be added to the controller automaton to constrain its behaviour and thus increase efficiency (see Section 5.5).

4.3 Correctness

For α to yield a *correct over-approximation*, every finite trace of the concrete system $\mathcal{A}$ (represented by a model of $\varphi(\mathcal{A})_k$, see Theorem 3.3) has to be reproducible in the abstract case, which is established by the following lemma (see [8] for a formal proof).

Lemma 4.2 (Abstraction by weakening) *The abstraction MO yields a conservative approximation, that means $\alpha(F)$ is weaker than F in the sense that the implication $F \rightarrow \alpha(F)$ is* valid *(true in all models).*

In order to relate our logical abstraction refinement approach to abstraction on TA, and emphasise the structural relationships, we prove a stronger correctness result than the one expressed by Lemma 4.2. For this, we use a homomorphic correspondence between concrete and abstract system [4].

Definition 4.3 [Homomorphism of traces] Let $\mathcal{A}$ and $\tilde{\mathcal{A}}$ be TA, with $\tilde{X} \subseteq X$, let $\gamma : S \cup \underline{A} \rightarrow \tilde{S} \cup \underline{\tilde{A}}$ a surjection. A function $h_T : Trace_{\mathcal{A}} \rightarrow Trace_{\tilde{\mathcal{A}}}$ is called a *homomorphism of traces* iff for each trace $t \in Trace_{\mathcal{A}}$, there is a trace $h(t) = \tilde{t} \in Trace_{\tilde{\mathcal{A}}}$ such that (a) for $i \geq 0$, the i-th configurations (s_i, ν_i) and $(\tilde{s}_i, \tilde{\nu}_i)$ in t and $\tilde{t}$, respectively, satisfy: $\gamma(s_i) = \tilde{s}_i$, and ν_i and $\tilde{\nu}_i$ agree on common variables, and (b) for $i \geq 1$, the $i-$th steps $(s_{i-1}, \nu_{i-1}) \xrightarrow{a_i} (s_i, \nu_i)$ and $(\tilde{s}_{i-1}, \tilde{\nu}_{i-1}) \xrightarrow{\tilde{a}_i} (\tilde{s}_i, \tilde{\nu}_i)$ satisfy: $\gamma(a_i) = \tilde{a}_i$ (accordingly for finite traces).

From an algebraic perspective, the stronger result even follows immediately from Lemma 4.2 as a key property, since α works locally and, thus, retains the formula structure of (9).

Theorem 4.4 (Correctness of abstraction) *MO, as defined in Def. 4.1, yields a correct over-approximation on trace sets.*

To prove the existence of a homomorphism between concrete and abstract traces (see [8] for details), the upper part of Fig. 7 is shown to be a commutative diagram, such that—by composition—the existence of h_T is a direct consequence. The idea of this proof is as follows: As α retains the form of (9), there is some automaton $\tilde{\mathcal{A}}$

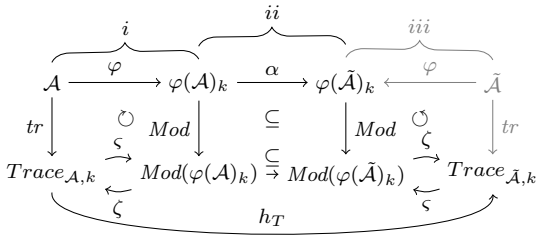


Fig. 7. Strong Correctness of Abstraction

of the same representation $\varphi(\tilde{\mathcal{A}})_k = \alpha(\varphi(\mathcal{A})_k)$ (up to logical equivalence). Thus, the subdiagrams of Fig. 7 marked (i) and (iii) commute according to Theorem 3.3. The subdiagram marked (ii) commutes according to Lemma 4.2, as every model of $\varphi(\mathcal{A})_k$ is a model of $\alpha(\varphi(\mathcal{A})_k)$. Hence, the whole diagram commutes.

4.4 Discussion

Thanks to the uniform nature of α, we generally do not have to distinguish between separate treatments of state abstraction, event abstraction, and clock abstraction. On a logical level, these are just arbitrary symbols that happen to represent different automata concepts. In particular, the algebraic view even permits to conclude from the strong correctness result that there is a corresponding effective abstraction technique on TA (as opposed to logical formulas) that produces $\tilde{\mathcal{A}}$ (the above proof of $\tilde{\mathcal{A}}$ is simple but non-constructive). Yet, proving that such a technique is correct (and even stating it) will be more complicated and much less uniform than what our embedding in logic has been able to express (Fig. 7 illustrates the interplay of logical abstractions and abstractions on automata).

Thus, we strike a good balance between universality and efficiency: every abstraction α satisfying Lemma 4.2 has already been proven correct in our framework, which makes it a powerful technique. Due to the purely syntactic definition, it is very efficient, though.

5 Abstraction Refinement

In this section, we present our general abstraction refinement methodology, following the general abstraction refinement paradigm [4]: (1) Generate the initial abstraction, (2) model check the abstract system, and, if required, (3) refine the abstraction (cf. Fig. 1). We discuss how to detect spurious counterexamples, and what kind of information can be derived from spurious counterexamples for finding an adequate refinement.

5.1 Generating Initial Abstractions

If there is no additional knowledge about the system, the initial abstraction simply removes all symbols contained in $\Phi(X) \cup X$ from $\varphi(\mathcal{A})$, and merges all symbols in S to a single one (we refer to [4] for improved techniques), thereby collapsing to a

single trivial state (accordingly for $\underline{A}$). Yet, the next iterations of refinement will quickly discover more relevant parameters.

5.2 Model Checking the Abstract System

After abstraction, $\alpha(\varphi(\mathcal{A}))_k \wedge \rho_k$ is checked for satisfiability. If it is unsatisfiable, the system is safe within bound k (Section 3.3 and 4.3).

If, instead, the SAT solver returns a satisfying model (which corresponds to a counterexample trace according to Theorem 3.3), let π the conjunction of variable assignments according to this model. This abstract counterexample needs to be *concretised*, i.e., translated back to the terminology of the concrete system, and checked for feasibility, which amounts to a satisfiability check of $\varphi(\mathcal{A})_k \wedge \rho_k \wedge \pi$. For states $s, s' \in S$ with $\gamma(s) = \gamma(s') = u$, we further add the concretising constraint $u \rightarrow s \vee s'$ to π, which expresses that $\mathcal{A}$ either is in state s or s' whenever π is in u (accordingly for events or more states).Hence, the single abstract counterexample trace corresponds to (possibly several) traces of $\mathcal{A}$ that are characterised by forming the disjunction of the preimage of γ for states and events. Observe that the clock constraints remain unchanged during this process, since the values for clocks are either unknown from the abstract counterexample (if dropped, i.e., in $\mathcal{AS}$) or unchanged. As π is highly restrictive (it singles out only one abstract path) the abstract counterexample guides the search through the concrete system with a very narrow focus and is highly efficient (Section 5.5).

If $\varphi(\mathcal{A})_k \wedge \rho_k \wedge \pi$ is satisfiable, this yields a concrete counterexample to the property. Otherwise, π is spurious and the abstraction has to be refined (see Section 5.3).

5.3 Refining Abstractions

We use *Craig interpolants* (e.g. [9]) provided by the SAT solver (FOCI) to identify ill-abstracted parameters. A Craig interpolant for an inconsistent pair of formulas (A, B) is a formula C that is implied by A (*prefix* of C), inconsistent with B (*suffix* of C) and refers only to common symbols of A and B. C is thus a joint over-approximation of A and an under-approximation of $\neg B$.

After stratifying $\varphi(\mathcal{A})_k$, π and ρ_k (i.e., aligning formulas along the unfolding depth k to which they belong), we derive an interpolant for every partition into prefix and suffix. By this, we obtain a sequence of strong interpolants (refer to [8] for details), such that there is a last interpolant $G \neq \texttt{false}$. By definition, the prefix of a false interpolant is unsatisfiable. Hence, the corresponding prefix of π represents a trace that is non-concretisable in $\mathcal{A}$. In case there is an interpolant $\tilde{G} = \texttt{true}$, we can even rule out the subtrace represented by those formulas between $\tilde{G}$ and G, as true interpolants express that there is no information whatsoever to carry over from the trace prefix up to this index. Furthermore, when P denotes the set of all symbols subject to abstraction, we know that at least one of the symbols in $\mathit{IA} \stackrel{def}{=} cont(G) \cap P$ has been inadequately abstracted.

Hence, there are two choices for refinement: (a) refine a symbol from IA, or (b) rule out the counterexample trace represented by the prefix of G by adding

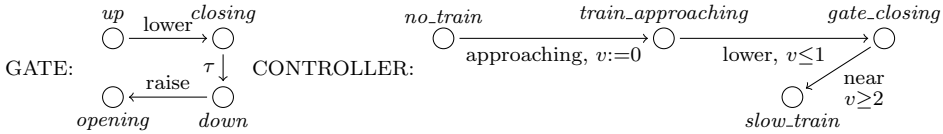


Fig. 8. Refinement: Running Example

a corresponding conjunction (CEGAR in [4]). Although (a) is reasonable (cf. Section 5.1), premature refinement with (a) slows down overall verification by collapsing to the concrete system too soon. Strategy (b), on the other hand, is very simple, but leads to a large number of spurious counterexamples as long as the essential parameters have been inadequately abstracted.

We have identified the following *fully automatic* heuristic as a compromise between these alternatives: after refining a parameter (a), a fixed number of traces (fractions of the unrolling depth k have turned out to be most promising) is ruled out by (b) before refining the next symbol according to (a). Further optimisations include keeping track of all counterexamples found so far and refine symbols in IA first that occur most frequently. Similarly, as a clean-up strategy, counterexamples added by (b) can be removed once their signature has been covered by subsequent refinements according to (a).

5.4 Running Example

For simplicity, let the map of merging γ be the identity, and let the set of omissions $\mathcal{AS}$ contain clock y (singleton). The parts of the GATE and CONTROLLER automaton depicted in Fig. 8 show this situation, where it is possible to reach state *opening*$\wedge$*slow_train* (and thus also the error state). Concretising this trace, however, is not possible, as within the corresponding part of Fig. 2, always $y \leq v$ holds (which avoids synchronising on *near* before going from *closing* to *down*), and the SAT solver returns a sequence of interpolants. We know that the last interpolant $G \neq \texttt{false}$ will contain clock y[7] (cf. Section 5.3: $\mathcal{AS}$ is singleton, thus only y can cause the spurious counterexample), and thus y has to be refined.

Note: when verifying larger systems, these do not collapse to the concrete system, but our approach correctly identifies the abstractable parameters.

5.5 Preliminary Experimental Results

Some preliminary experimental results are depicted in Tab. 1. Starting from the initial abstraction described in Section 5.1, Tab. 1 shows that our overall abstraction refinement technique is very quick since the predominant run-time is spent on (external) SAT solver calls ($\geq$83%). The efficiency gains of our abstraction refinement framework are pointed out by the observation that the last iteration—i.e., when all relevant parameters (all but clock x) have been identified by our refinement—needs a disproportional amount of time. Hence, the first runs for "abstraction discovery"

[7] For example, the interpolant could contain the constraint $(\mathtt{z_t} - \mathtt{y_t} \geq 2)$, expressing the fact that clock y has to be greater than 2 in step t for this trace to be concretisable.

Example in	Strategy [8]	Complete Run	Hereof: SAT solver calls	Hereof: Last Iteration
Section 2	$1/2k$	16 : 889	14 : 594	8 : 163
Section 4.2	$1/2k$	16 : 631	14 : 334	7 : 866
Section 2	$1/3k$	15 : 651	13 : 081	5 : 445
Section 4.2	$1/3k$	16 : 957	14 : 439	6 : 423
Section 2	$1/4k$	11 : 312	9 : 665	6 : 665
Section 4.2	$1/4k$	11 : 063	9 : 451	6 : 279

Table 1
Preliminary Experimental Results for TGC example

are quick in comparison. Strengthening α using inferred information (Section 4.2) leads to at least a slightly improved performance. Tab. 1 also shows that efficienty depends on the chosen heuristic.

6 Conclusions and Future Work

Within the context of this paper, we have presented a SAT-based approach for abstraction refinement model checking of parallel systems of TA.

We have defined an embedding of bounded model checking for systems of TA into propositional logic with linear arithmetic, and introduced a uniform logic-based abstraction for clocks, states, and events. This logical representation directly benefits from state-of-the-art techniques of SAT solvers, and allows a linear-size representation of parallel composition.

In addition to having proven correctness of representation and abstraction, we carefully identify the algebraic and logical principles underlying our abstraction refinement approach. We expect those structural relationships to provide a modular framework for generalising aspects of our work to other scenarios (like hybrid automata) without having to start from scratch.

Besides those generalisations, future work includes performance comparisons of our implementation on case studies, and an analysis of the effect of choosing a logarithmic encoding for states (state abstraction is slightly more involved in that case, though). The overall performance might be improved using a SAT solver that is tailored towards bounded model checking with pseudo-boolean constraints or isomorphy inference. Moreover, we intend to examine different settings for balancing counter-example guided abstraction refinement versus refinement based on Craig interpolants.

As a general philosophy, we propose to extend our investigation of the algebraic relationships underlying current research about model checking in order to (a) properly identify the structural principles, and (b) build a stable framework for generalisations to more advanced verification problems, e.g. [5].

Acknowledgement

Earlier versions of this work have benefited from discussions with members of AVACS, particularly Ernst-Rüdiger Olderog, Martin Fränzle, Henning Dierks, An-

[8] Strategy denotes the number of traces in relation to unrolling depth k to be ruled out before refining the next parameter

dreas Podelski and Andrey Rybalchenko.

References

[1] Alur, R., *Timed automata.*, in: N. Halbwachs and D. Peled, editors, *CAV*, LNCS **1633** (1999), pp. 8–22.

[2] Audemard, G., A. Cimatti, A. Kornilowicz and R. Sebastiani, *Bounded model checking for timed systems.*, in: D. Peled and M. Y. Vardi, editors, *FORTE*, LNCS **2529** (2002), pp. 243–259.

[3] Clarke, E., A. Biere, R. Raimi and Y. Zhu, *Bounded model checking using satisfiability solving.*, Formal Methods in System Design **19** (2001), pp. 7–34.

[4] Clarke, E., O. Grumberg, S. Jha, Y. Lu and H. Veith, *Counterexample-guided abstraction refinement for symbolic model checking.*, J. ACM **50** (2003).

[5] Henzinger, T., *The theory of hybrid automata.*, in: *LICS*, 1996, pp. 278–292.

[6] Henzinger, T., R. Jhala, R. Majumdar and K. McMillan, *Abstractions from proofs.*, in: N. D. Jones and X. Leroy, editors, *POPL* (2004), pp. 232–244.

[7] Jhala, R. and K. McMillan, *Interpolant-based transition relation approximation.*, in: *CAV*, 2005, pp. 39–51.

[8] Kemper, S., "SAT-based Verification for Abstraction Refinement," Master's thesis, University of Oldenburg (2006).
URL http://csd.informatik.uni-oldenburg.de/~skript/pub/diplom/kemper06.pdf

[9] McMillan, K., *An interpolating theorem prover.*, Theor. Comput. Sci. **345** (2005), pp. 101–121.

[10] Moskewicz, M., C. Madigan, Y. Zhao, L. Zhang and S. Malik, *Chaff: Engineering an efficient SAT solver.*, in: *DAC*, 2001, pp. 530–535.

[11] Platzer, A., *Towards a hybrid dynamic logic for hybrid dynamic systems*, in: P. Blackburn, T. Bolander, T. Braüner, V. de Paiva and J. Villadsen, editors, *Proc., LICS International Workshop on Hybrid Logic, Seattle, USA*, ENTCS, 2006.

[12] Taylor, P., "Practical Foundations of Mathematics," Cambridge University Press, 1999.